# A metaheuristic based on fusion and fission for partitioning problems

Charles-Edmond Bichot
Laboratoire d'Optimisation Globale (LOG)
DSNA-DTI — ENAC
8, avenue Edouard Belin 31 000 Toulouse, FRANCE
bichot@recherche.enac.fr

## Abstract

*Metaheuristics are very useful methods because they can find (approximate) solutions of a great variety of problems. One of them, which interests us, is graph partitioning. We present a new metaheuristic based on nuclear fusion and fission of atoms. This metaheuristic, called fusion fission, is compared to other classical algorithms. First, we present spectral and multilevel algorithms which are used to solve partitioning problems. Secondly, we present two metaheuristics applied to partitioning problems : simulated annealing and ant colony algorithms. We will show that fusion fission gives good results, compared to the other algorithms. We demonstrate on a problem of Air Traffic Control that metaheuristics methods can give better results than specific methods.*

## 1 The partitioning problem

Graph partitioning is a fundamental problem for many scientists. This problem arises in many graphs applications and consists in dividing the vertices into several sets of roughly equal "size" in such a way that the weight of edges between sets is as small as possible. A classical application of graph partitioning is parallel computing, to reduce the communication between processors. But applications of graph partitioning include also VLSI design, data clustering, image segmentation, and mesh partitioning of a 2D surface of an airfoil. A new partitioning problem consists of a new organization of the European airspace. This is the Functional Airspace Block Optimized Process (FABOP) project. We will see more precisely this subject in section 5.

It is known that finding an optimal partition of a graph is NP-complete [11]. This problem inspired a great number of methods and heuristics. Some metaheuristics have already been used to partition graphs, like genetic algorithms [28, 12] or ant-like agents [22, 23]. Therefore, for all of these tools, we are looking for a near optimal partition in reasonable time. Because minimum cut algorithms were well studied [20, 9, 7, 3, 27], most partitioning methods use recursive bisections. But these methods often provide a partition which is far from optimal [26], regarding the minimization of the sum of the weight of edge cuts. Conversely, spectral graph partitioning methods [24, 13] and multilevel partitioning algorithms [17] produce good partitions.

We will now describe more formally the problem of graph partitioning. Given a graph $G = (V, E)$ with vertex set $V$ and edge set $E$, and a weight function $w$ on edges (this function can represent the number of edges crossing two vertices), $\forall\ e \in E, w(e) \geq 0$. Let $P_k(G)$ be a partition of $G$ into k non-empty subgraphs $V_1, \ldots, V_k$ with $\forall\ i, j = 1 \ldots k, i \neq j, V_i \cap V_j = \varnothing$ and $\bigcup_{i=1}^{k} V_i = V$. Let $A \in P_k(G)$, $V - A = \{u \in V, u \notin A\}$, we define $cut(A, V - A) = \sum_{u \in A, v \in V - A} w(u, v)$ and $W(A) = \sum_{u \in A, v \in A} w(u, v)$.

Depending on the partitioning instance of the problem, there are different objective functions. The first and simplest is the *cut* function, which corresponds to the minimum-cut problem, *ie.* minimizing the sum of weight of edges across sets :

$$Cut(P_k(G)) = \sum_{A \in P_k(G)} cut(A, V - A)$$

The optimal $k$-partition of the graph, regarding the minimum-cut problem, is the one minimizing this *cut* function. An other weight function is the normalized cut [25]. The goal of this criterion is to minimize cuts between sets, and to maximize for each set, the sum of the weight of edges which have at least one vertex in this set :

$$Ncut(P_k(G)) = \sum_{A \in P_k(G)} \frac{cut(A, V - A)}{assoc(A, V)}$$



1-4244-0054-6/06/$20.00 

where $assoc(A,V) = cut(A,V-A)+W(A)$. At last we introduce the min-max cut function or $Mcut$ [4], which aims to minimize cut between sets and to maximize for each set the sum of the weight of edges for which the two vertices are in this set :

$$Mcut(P_k(G)) = \sum_{A \in P_k(G)} \frac{cut(A, V-A)}{W(A)}$$

As we can see, graph partitioning has been studied for various criteria, with distinct methods. Certainly, the most adaptive method is metaheuristic. This method can easily change of goals, *ie.* criteria. On the opposite side, spectral, multilevel or heuristics methods are designed for one criterion precisely. But it is interesting to compare for each method, with its criterion, the results with metaheuristic's results. We rapidly introduce in section 2 the spectral and multilevel graph partitioning methods. Next, we introduce, in section 3, classical metaheuristics which are simulated annealing and ants colony. In section 4, we present a new heuristic based on nuclear fusion and fission. We then present a new application of graph-partitioning, based on a problem of air traffic control, in section 5. And finally, we present in section 6 the results of our tests applied to air traffic control for the different methods used.

## 2 Spectral and multilevel graph partitioning methods

In this section, we present spectral and multilevel graph partitioning methods. These algorithms have been implemented and integrated into tools. The most populars of these tools are the Chaco software, developed by Hendrickson and Leland [16], and the Metis software, developed by Karypis and Kumar [18].

### 2.1 Spectral graph partitioning method

Spectral methods use eigenvectors of a matrix constructed from the graph to decide how to partition the graph. The most widely known methods use the Fielder linear order [5, 10, 24]. Other methods exist [29], but their use is marginal.

We first explain this method for a bisection of $G = (V, E)$ into two sets $A$ and $B$. Let $W$ be the matrix of edge's weights. $\forall u \in A$, we define $d(u) = \sum_{v \in V} w(u,v)$. $D$ is the diagonal matrix with $d$ on its diagonal. If we use a vector $x$ of indicator variables $x_u = \{-1, 1\}$ depending on $u \in A$ or $B$, we have :

$$Cut(A,B) = \sum_{(u,v) \in E} \frac{(x_u - x_v)^2}{2} w(u,v) = x^T(D-W)x$$

If we relax each $x_u$ from $\{-1, 1\}$ to continuous value in $[-1, 1]$, minimizing $Cut(A, B)$ is equivalent to solve the eigensystem $(D-W)x = \lambda x$. We are interested by the eigenvectors of the Laplacian matrix $L = D - W$. The first and trivial eigenvector $x_1 = (1, \ldots, 1)^T$ is associated with $\lambda_1 = 0$. The second eigenvector, called the Fielder vector, is the solution. For the $Ncut$ criterium, we must solve the generalized eigenvalue system $(D-W)x = \lambda Dx$. And for the $Mcut$ criterium, $(D-W)x = \lambda Wx$. The new problem is to solve the eigenvector system. The Lanczos method is probably the most known method to solve it. It is efficient for graphs no larger than 10,000 vertices. But there exist also the RQI/Symmlq method [16].

To simultaneously cut the graph into $2^n$ sets, we can use the $n$ top eigenvectors in the Fielder order (after removing the first eigenvector). We use its as $n$ indicator vectors for the $2^n$ partitions. As we say, the first eigenvector gives a bisection of the graph. The second eigenvector gives a bisection of the graph too. If we keep the bisection given by the first eigenvector, we have now a quadrisection of the graph. The third eigenvector gives an octasection of the graph and so on. Like recursive bisections, this method is not appropriated for partitioning into $k \neq 2^n$ sets. For further details of this method, see [15, 25].

### 2.2 Multilevel graph partitioning method

There are different multilevel graph partitioning methods. But the scheme of these methods is to construct a coarse graph, then to partition this graph into k partitions, and to uncoarsen the graph with local refinement. A multilevel method, due to Karypis and Kumar, is explained in [19] and used in the Metis software. We are explain next the method of Hendrickson and Leland [17], which is used in the Chaco software.

As we have explained above, there are three steps in the multilevel method.

- Constructing a coarse graph. The aim is to make a contraction of a large number of edges that are well dispersed throughout the graph. In this step two vertices ($a$ and $b$) joined by an edge are merged. The new vertex ($c$) wheigth value is the sum of the weights of $a$ and $b$. Each edge of adjacent vertices of $a$ and/or $b$, have a new weight which is the sum of the weight between the adjacent vertex and the two vertices $a$ and/or $b$.

- Partitioning the graph. In this step, different partitioning algorithms can be used. Hendrickson and Leland used a spectral method which uses the eigenvectors of the Laplacian matrix.

- Uncoarsening the graph. Because each vertex in a coarse graph is simply the union of vertices from a larger graph, this step is trivial. But the best partition of the coarse graph may not be optimal for its uncoarsen counterpart. Therefore a local refinement to the uncoarsened partition can be applied.

### 2.3 Local refinement methods

Because spectral and multilevel methods are not locally optimal, a local refinement algorithm can be incorporated in this method. Local refinement methods are usually generalizations of the bisection algorithm of Kernighan and Lin [20] with the linear time implementation of Fiduccia and Mattheyeses [9]. With local refinement, results are generally 10 to 30% better [16].

## 3 Two classical metaheuristic methods

### 3.1 Simulated annealing

Simulated annealing is the oldest of all metaheuristics. The origins of the method are in statistical mechanics (Metropolis algorithm). The algorithm was first presented in [21]. The fundamental idea is to allow moves resulting in solutions of worse quality than the current solution (uphill moves) in order to escape from local minima. The probability of doing such a move decreases during the search. Annealing is a term from metallurgy. If a metal is heated at a very high temperature, atoms are misplaced and moved quickly. If they are cooled very slowly, they settle into patterns and structures, making the metal much stronger than before.

The algorithm starts with an initial solution (constructed randomly or by an heuristic) corresponding to a very high temperature $T$. Let $e$ be the objective function, also called "energy" function, and $D$ the function which decreases the temperature (cooling schedule). Until a termination condition is satisfied, the algorithm repeats:

Let $s$ be the current solution. A solution $s'$ in the neighborhood of $s$, is chosen by the perturbation function. This solution is accepted if $e(s') \leq e(s)$ and, also if the Boltzmann probability $\exp\frac{e(s)-e(s')}{T}$ is larger than a random number in $[0, 1]$. Then, if the algorithm reaches the equilibrium condition, the temperature $T$ is decreased to $D(T)$. When the freezing point is reached (stopping criterion), the best solution is returned.

Simulated annealing converge under certain conditions to the optimum solution of the problem, but in infinite time. A great advantage of simulated annealing related to other heuristics is its adaptivity on different problems and his simplicity of implementation. Previous work has already adapted simulated annealing to graph partitioning [7]. Our adaptation differs from this one. The perturbation function chooses randomly a vertex in $V$. This vertex is moved to an another partition. If the temperature is high, this partition is the lowest partition regarding the sum of edges weight which are entirely inside partitions, otherwise it is randomly chosen between connected partitions. Thus, connectivity between sectors is not forced. Equilibrium condition is reached when a fixed number of solutions are refused. The cooling schedule is $D(T) = T * \frac{t_{max}-t_{min}}{t_{max}}$ where $t_{max}, t_{min}$ are the maximal and minimal temperature. Stopping criterion is $T \leq t_{min}$.

### 3.2 Ant colony

Ant colony optimization is inspired by the foraging behavior of real ants. This metaheuristic proposed by Dorigo is explained in [6]. The algorithm uses the ability of ants to find the shortest path between food source and their nest. While walking from food sources to the nest and *vice versa*, ants leave a substance called pheromone on the ground. When they decide about a direction to go, they choose with higher probability paths that are marked by stronger pheromone concentrations.

The ant colony algorithm is based on three (one optional) steps. While a termination condition is not satisfied, the three steps are executed, but not necessary in the order we propose.

- The first step is the motion of ants. Ants move through nodes of the graph $G$ by applying a stochastic local decision policy which uses pheromone values and a local heuristic. While moving, the ant keeps in memory the path used in the graph.

- During the second, pheromones are updated. Ants always update the pheromone trails they are using. But if a path leads to "food" (a local solution), the ant can update backward the path it used by using its memory. Finally, like real pheromones, pheromone trail intensity decreases over time (to avoid convergence into a sub-optimal region).

- The last step is optional. It is used to implement centralized actions which cannot be performed by single ants.

Our adaptation of ant colony algorithm to the $k$-partitioning problem uses $k$ colonies, each one representing a partition of the graph. These colonies are in competition for food. Each colony has its own pheromones, *ie.* an ant can only distinguish pheromones of its colony. The weight of vertices (usually the sum of the weights of each connected edges) corresponds to ant's food. Ants put down pheromones on edges. A vertex is owned by a colony if the sum of its pheromones on adjacent edges is greater than for other colonies. A local heuristic forces ants to explore edges which have no pheromone. Ants can go where they want, so ants from different colonies can be in the same vertex at the same step. Thus, the connectivity of sets is not forced. It is important to observe that if connected sets often produced best results, we should not force this connectivity.

To conclude, our approach of $k$-partitioning with ant colony is very different than preceding works [22, 23]. Further details of our approach are explain in previous work [2].

## 4 A new metaheuristic method based on fusion and fission

The $k$-partitioning problem inspires our method. A very simple process of $k$-partitioning is to successively put one vertex of a set into a different set, and to see if the result is better. A vertex is simply a particle of a set, and the set containing all sets is a structured world. We can make an analogy between a vertex and a nucleon, a set and an atom, and the set containing all sets and molecule. Making a structured molecule is just moving nucleons from atoms to atoms.

All partitioning methods solve problems for a constant number $k$ of partitions. But our work on air traffic control showed us that we can search for $k$, $k-1$ or $k+1$ partitions. Moreover, sometimes, a $k-1$ partition is greatly better than a $k$ partition, regarding the objective function. Our new method is based on the change of the number of partitions. Nevertheless, we must know an approximate value for $k$.

Because of the limited space available, we explain this method through the $k$-partitioning problem. For more details see [2].

### 4.1 Introduction of the method

Like other metaheuristics, this method infused on nature behaviour. Fusion and fission are two processes which organize atoms. Fusion is the process which merges two atoms in one. On the contrary, fission is the process which breaks atoms in two parts. The method consists in merging and breaking atoms successively. Then, applied to the partitioning problem, the number of partitions changes over time.

Because objectives functions of section 1 are computed for a constant $k$ number of partitions, the fusion fission method uses an scaling function after the calculation of the objective function. This scaling function is used to produce the curve of binding energy. Binding energy is the energy required to break an atom. Considering the number of nucleons per atom, binding energy of light elements increases fast; there is afterwards a region of stability, and then, the binding energy of big elements decreases slowly. The objective function generally return lower results for a graph with few partitions than a lot (results is the smallest when there is no partition). Graphs which have different number of partitions but the same quality of partitioning, regarding the objective function only, return different energies. After the scaling function applied to the objective function, energies are the same for the same quality of partitioning.

The process of choice between fusion and fission is a simple heuristic which guides our partitioning method around the number of $k$ partitions, regarding the energy of the partitioning (the result returned after the upstream function and the objective function).

Temperature has a highly significant role in fusion and fission process. The higher the temperature is, the easier the fusion of big atoms is and the easier the fission of small atoms is. Thus, it is important to include temperature in our method.

In nature, fusion and fission obey to laws. Some fissions in two atoms leave nucleons alone (especially when the atom is big). In the same way, fusion of two atoms can make a new atom and eject one or more nucleons. The algorithm includes these laws, but with a memory which updates laws (if the law gives a better solution, the process is enforced, else it is weakened). Each law depends on the number of nucleons of, the atom(s). The number of laws is twice the number of vertices (one for fusion plus one for fission). Each law is composed of four probabilities (less if the sum of nucleons is lower). The first one is the probability to eject no nucleon, the second to eject one nucleon and so on. The sum of these four probabilities must be one. If the number chosen gives good results (the energy of the new partition is lower), we add to its probability an input value and remove to the other probabilities the third of this input value. Of course, each probability must be strictly in $[0, 1]$.

### 4.2 Algorithms of fusion fission

Because of the particular approach of our method, we have to initialize the process. The goal of this initialization is to construct a (near) $k$-partition of the problem. This initialization is a simplification of the core algorithm, so we will discuss it later.

We use distance between two atoms (partitions). This is the inverse of the sum of the weights of connected edges between these atoms (partitions). The distance is infinity if the partitions are not connected.

Let's explain the core algorithm (Algorithm 1). It start with a (near) $k$-partition of the problem which has high temperature, and runs step by step. Let $S_i$ be the partition of the graph at step number $i$. An atom is randomly selected by *choose_atom*. A function of probability, *random*, decides if this atom is to be cut or joined with an another atom. A law is selected. Depending on the result of *random*:

- Fusion. A second partition is selected according to its size, its distance to the first one, and temperature. The two partitions are joined together. According to the law, some nucleons are ejected. These nucleons are incorporated into different atoms connected with them.
- Fission. A partition is cut in two sets. According to the law, some nucleons are ejected. If temperature is high, these nucleons can produce another simple fission, with no nucleon ejected. Atoms connected to these nucleons are chosen for these fissions. Else, if the temperature is low, this nucleons are incorporated into atoms.

Next, the law is updated and the temperature is decreased. If the temperature is too low, the algorithm is initialized with the best partition found and the higher temperature. Else, (even if energy is higher) a new step begins with the current partition.

As explained earlier, the initialization is a simplification of algorithm 1. Initialization starts with a graph where each partition is a vertex. The number of partitions and the number of vertices are the same. The energy of such a graph is maximal. The first goal of initialization is to group vertices, to obtain a near $k$-partition. The initialization process is described in algorithm 2. Compared to the core algorithm, we remove temperature, fission produced by nucleons, and use a different heuristic to choose between fusion and fission.

### 4.3 Functions used by fusion fission

The fusion fission algorithm uses a function to decrease temperature, $decrease(t)$ and a function to

**Algorithm 1** Fusion / Fission

Contractions: nfusion(nfission) from nucleon fusion(fission), cpart(npart) from current(new) partition, n(ln) from nucleon(list of), t from temperature, nlaws from new laws.

```
cpart ⇐ best_part;
t ⇐ max_t;
while Stop condition do
  atom ⇐ choose_atom(cpart);
  if random(atom, cpart) is fusion then
    (npart, ln) ⇐ fusion(atom, cpart, laws);
    for all n ∈ ln do
      npart ⇐ nfusion(n, npart, laws);
  else
    (npart, ln) ⇐ fission(atom, cpart, laws);
    for all n ∈ ln do
      if high_energy(n, t) then
        npart ⇐ nfission(n, npart, laws);
      else
        npart ⇐ nfusion(n, npart, laws);
  new_laws ⇐ update(laws, t);
  new_t ⇐ decrease(t);
  if Energy(npart) < Energy(cpart) and
  Energy(npart) < Energy(best_part) then
    best_part ⇐ npart;
  if low_temperature(t) then
    cpart ⇐ best_part; t ⇐ max_t;
  else
    cpart ⇐ npart; t ⇐ new_t;
```

**Algorithm 2** Initialization

Contractions: same than algorithm 1

```
while Stop condition do
  atom ⇐ choose_atom(cpart, laws);
  if random(atom, cpart) is fusion then
    (npart, ln, nlaws) ⇐
                        fusion(atom, cpart, laws);
  else
    (npart, ln, nlaws) ⇐
                        fission(atom, cpart, laws);
  for all n ∈ ln do
    (npart, nlaws) ⇐ nfusion(n, npart, nlaws);
  cpart ⇐ npart; t ⇐ new_t;
```

choose between fusion and fission, $choice(x)$.

$$decrease(t) = t\ \frac{t_{max} - t_{min}}{nbt}$$

The temperature will decrease $nbt$ time befor reaching $t_{min}$.

The choice function depends on the number $x$ of nucleons of the atom chosen (the atom from which we choose fusion or fission). We define $n = \frac{nbv}{k}$, with $nbv$ the vertex number of the graph, and the function of temperature $\alpha(t) = k\ \frac{t_{max}-t}{t_{max}-t_{min}} + r$, with $t_{max}, t_{min}$ the maximal and minimal temperature, and $k, r$ adjusted by the user.

$$choice(x) = \begin{cases} 1 & if\ x > n + \frac{1}{2\alpha(t)} \\ 0 & if\ x < n - \frac{1}{2\alpha(t)} \\ \alpha(t) * (x - n) + \frac{1}{2} & else \end{cases}$$

### 4.4 Percolation to cut atoms

We use a process which is like percolation to cut partitions. This process is used by fission to cut atoms. A percolation is a liquid flow which goes through a porous substance. The liquid starts on a place, and then drips gradually all over this place.

We define $k$ different vertices $c_i$, from which $k$ different colored liquids can start. Each color represents a partition $P_i$. The algorithm progresses step by step. A vertex $v$ is added to a partition $P_i$ (it is colored) if $\forall j, bond(v, P_i) > bond(v, P_j)$, with :

$$bond(v, P_i) = \sum_{e\ \in\ path\ from\ c_i\ to\ v} \frac{w(e)}{2^d}$$

where $d$ is the number of vertex between $e$ and $c_n$. All bonds are recomputed at each step. The distance to the center of a partition is represented by the division by $2^d$. For a vertex $v$, for each connected vertex which already has bonds, a new bond is calculated with the weight between these two vertices. The lowest bond and the corresponding path are assigned to $v$. This path is not always the shortest, and can change during the process. The algorithm stops when no vertex moves to an other partition.

This process is fast and can partition a graph into $k$ partitions. We use it to initialize ant colony, simulated annealing, and we use it during fission to cut partitions into two.

## 5 Application on Air Traffic Control

"The primary purpose of the Air Traffic Control (ATC) system is to prevent a collision between aircraft operating in the system and to organize and expedite the flow of traffic" (Federal Aviation Administration [8]). The first objective of ATC is safety, the second is efficiency. The FABOP project is a study in the "strategic" level of ATC, *ie.* our methods are applied long before any tactical control of aircraft. However, we are working for safety and efficiency.

Each air traffic controller supervises a limited space, called an air traffic sector. Controllers have qualifications to work only on a set of sectors. These sets are sometimes called functional airspace blocks. The FABOP project consists in cuting the European airspace into blocks. Currently blocks almost never cross countries border. We study a new organization of blocks only based on flows of aircraft and not on borders.

Because "it is well known that controller-controller coordination is easier and more effective inside an ATC unit (a block) than between ATC units" [14], we search to maximize flows of aircraft inside blocks and minimize flows of aircraft between blocks. The graph vertices are air traffic sectors and edges are flows of aircraft between sectors. The air traffic problem is a $k$-partitioning problem where $k$ is the number of functional airspace blocks into which we want to cut the European airspace. Regarding our objectives, the appropriate objective function to use is $Mcut$.

## 6 Results

We applied all these algorithms to our problem. The graph we want to cut is composed of 762 vertices and $3,165$ edges. This is the total number of sectors of what we have defined as the "country core area" [1], which is composed of Germany, France, United Kingdom, Switzerland, Belgium, Netherlands, Austria, Spain, Denmark, Luxembourg and Italy, which is the set of countries with the highest flows of aircraft in Europe.

We compare partitioning results, in table 1, of the three metaheuristics we have adapted, simulated annealing, ant colony and fusion fission, with results of spectral and multilevel graph partitioning methods, thanks to the Chaco software [16]. We use contractions in table 1: bi for bisection, oct for octasection, KL for Kernighan-Lin algorithm, Lanc for Lanczos, and RQI for RQI/Symmlq. Results are solutions of a 32-partition of the European sky.

Regarding the number of input parameters, simulated annealing is the simplest algorithm with one parameters to change, $t_{max}$ (we put $t_{min} = 0$). Percolation with the set of $k$ initial vertices is slightly more complicated. Ant colony has four parameters. The fusion fission algorithm has five parameters, $t_{max}$, $t_{min}$

**Table 1. Comparisons between algorithms. $Cut$ results are divided by 1 000**

| Method | $Cut$ | $Ncut$ | $Mcut$ |
|---|---|---|---|
| Linear (Bi) | 274.2 | 30.12 | 2300.85 |
| Linear (Bi, KL) | 210.4 | 23.35 | 89.09 |
| Linear (Oct, KL) | 216.5 | 23.97 | 105.16 |
| Spectral (Lanc, Bi) | 202.0 | 22.62 | 81.38 |
| Spectral (Lanc, Bi, KL) | 202.7 | 22.62 | 120.29 |
| Spectral (Lanc, Oct) | 201.0 | 22.56 | 89.89 |
| Spectral (Lanc, Oct, KL) | 203.1 | 22.88 | 88.18 |
| Spectral (RQI, Bi) | 203.2 | 22.58 | 79.58 |
| Spectral (RQI, Bi, KL) | 203.0 | 22.47 | 77.80 |
| Spectral (RQI, Oct) | 201.6 | 22.47 | 78.02 |
| Spectral (RQI, Oct, KL) | 202.4 | 22.31 | 75.45 |
| Multilevel (Bi) | 202.1 | 22.42 | 76.93 |
| Multilevel (Oct) | 201.7 | 22.49 | 78.84 |
| Percolation | 213.7 | 23.72 | 96.87 |
| Simulated annealing | 203.9 | 22.34 | 74.44 |
| Ant colony | 203.3 | 22.30 | 74.22 |
| Fusion Fission | 198.0 | 21.83 | 69.03 |

and $nbr$ for the temperature, $k$ and $r$ in $\alpha(t)$ for the choice function. We are surprised by the number of parameters of the Chaco software. Thus, it is difficult to obtain the best result. And this is why we present many results (linear, spectral and multilevel) from Chaco in table 1. For spectral and multilevel results, we use the REFINE_PARTITION parameter which increases considerably the quality of results. It is thus as difficult to choose parameters for spectral and multilevel algorithm, as for metaheuristics.

The objective function which interests us the most is $Mcut$ as we say in section 5. We can notice that best results are obtained for metaheuristics, first fusion fission, then with approximately same results, ant colony and simulated annealing. Later come spectral, multilevel, and last percolation. Considering $Ncut$, the classification is a little different. Fusion fission stays first, but spectral method with RQI/Symmlq, octasection and KL refinement, is second, then the other metaheuristics. Considering $Cut$, fusion fission is always first, but spectral and multilevel are before the other metaheuristics.

We must keep in mind that spectral and multilevel algorithms can only cut into $k = 2^n$ partitions, when metaheuristics can cut into all natural $k$ partitions. And, if fusion fission returns a 32-partition, it returns good solutions from 27 to 38 partitions.

Spectral and multilevel partitioning algorithms are much faster than metaheuristics. These algorithms

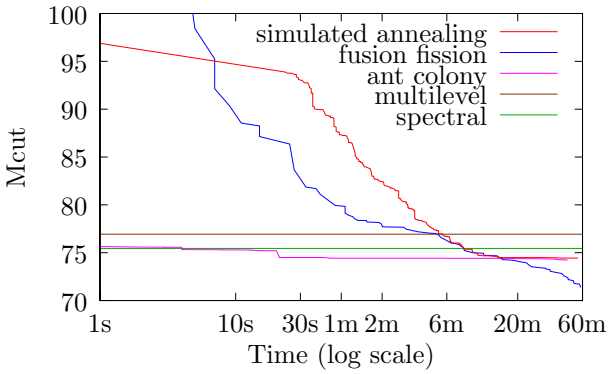


**Figure 1. Running time of the metaheuristics. Energies are compared to best spectral and multilevel cuts.**

take only a few seconds to compute, while the others can run infinitely. Figure 6 represents the time use by these algorithms on an Intel Pentium 4 CPU 3 GHz. As we can see, the algorithm of fusion fission starts with the worst initialization. Ant colony and simulated annealing start with the result of percolation. But ant colony loses 22% of energy in less than a second. Thus ant colony is the fastest of the three metaheuristics. Regarding results, the best is fusion fission.

## Conclusion

We present a lot of different algorithms: spectral and multilevel algorithms which are specific tools for graph partitioning, an heuristic adapted to graph partitioning (pecolation), and three metaheuristics: simulated annealing, ant colony and a new one, fusion fission. We applied all of these algorithms to an air traffic problem which is a $k$-partitioning problem. Results show that, if specific tools are faster than the others, metaheuristics give better results. It is important to notice that metaheuristics are not only theoretical tools, but can be successfully used dealing with real problems.

We present a new metaheuristic, named fusion fission, which gives the best results of all of these algorithms for our graph partitioning problem. This algorithm can be customized, especially throught choice function. Other choice functions not presented here give better results, but are much more complicated.

## References

[1] C.-E. Bichot and J.-M. Alliot. A theoretical approach to defining the european core area. Technical report, ENAC - LOG, 2005.

[2] C.-E. Bichot, J.-M. Alliot, N. Durand, and P. Brisset. Optimisation par fusion et fission. application au problème du découpage aérien européen. *Journal Européen des Systèmes Automatisés*, 38(9-10):1141–1173, 2004.

[3] C. Chekuri, A. V. Goldberg, D. R. Karger, M. S. Levine, and C. Stein. Experimental study of minimum cut algorithms. In *Symposium on Discrete Algorithms*, pages 324–333, 1997.

[4] C. H. Q. Ding, X. He, H. Zha, M. Gu, and H. D. Simon. A min-max cut algorithm for graph partitioning and data clustering. In *Proceedings of ICDM 2001*, pages 107–114, 2001.

[5] W. E. Donath and A. J. Homan. Lower bounds for the partitioning of graphs. *IBM J. of Research and Development*, 17:420–425, 1973.

[6] M. Dorigo, V. Maniezzo, and A. Colorni. The ant system: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics - Part B*, 26(1):29–41, 1996.

[7] F. Ercal, J. Ramanujam, and P. Sadayappan. Task allocation onto a hypercube by recursive mincut bipartitioning. *Journal of Parallel and Distributed Computing*, 10(1):35–44, 1990.

[8] Federal Aviation Administration (U.S. Department of Transportation). *Air Traffic ControL : FAA Order 7110.65K*, July 1997.

[9] C. M. Fiduccia and R. M. Mattheyses. A linear time heuristic for improving network partitions. In *19th ACM Design Automation Conf.*, pages 175–181, 1982.

[10] M. Fielder. A property of eigenvectors of non-negative symmetric matrices and its application to graph theory. *Czechoslovak Math. J.*, 25:619–633, 1975.

[11] M. Garey, D. Johnson, and L. Stockmeyer. Some simplified np-complete graph problems. *Theoretical Computer Science*, 1(3):237–267, 1976.

[12] W. A. Greene. Genetic algorithms for partitioning sets. *International Journal on Artificial Intelligence Tools*, 10(1-2):225–241, 2001.

[13] L. Hagen and A. Kahng. New spectral methods for ratio cut partitioning and clustering. *IEEE Transactions on Computer-Aided Design*, 11(9):1074–1086, 1992.

[14] A. Hallgren. Restructuring european airspace: functional airspace blocks. *Skyway*, pages 20–22, autumn 2005.

[15] B. Hendrickson and R. Leland. Multidimensional spectral load balancing. Technical Report SAND93-0074, Sandia National Laboratories, 1993.

[16] B. Hendrickson and R. Leland. *The Chaco user's guide*. Sandia National Laboratories, 2 edition, 1994.

[17] B. Hendrickson and R. Leland. A multi-level algorithm for partitioning graphs. In *Supercomputing*, 1995.

[18] G. Karypis and V. Kumar. *MeTis: Unstrctured Graph Partitioning and Sparse Matrix Ordering System*, 2 edition, 1995.

[19] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20(1):359–392, 1998.

[20] B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *The Bell System Technical Journal*, 49(2):291–307, 1970.

[21] S. Kirkpatrick, C. Gelatt, and M. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, May 1983.

[22] P. Kuntz, P. Layzell, and D. Snyers. A colony of ant-like agents for partitioning in vlsi technology. In *the Fourth European Conference on Artificial Life*, pages 417–424. MIT Press, 1997.

[23] A. E. Langham and P. W. Grant. A multilevel k-way partitioning algorithm for finite element meshes using competing ant colonies. In *the Genetic and Evolutionary Computation Conf.*, volume 2, pages 1602–1608, Orlando, Florida, USA, 1999.

[24] A. Pothen, H. D. Simon, and K.-P. Liou. Partitioning sparse matrices with eigenvectors of graphs. *SIAM J. Matrix Anal. Appl.*, 11(3):430–452, 1990.

[25] J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888–905, 2000.

[26] H. D. Simon and S.-H. Teng. How good is recursive bisection ? *SIAM Journal on Scientific Computing*, 18(5):1436–1445, 1997.

[27] M. Stoer and F. Wagner. A simple min-cut algorithm. *Journal of the ACM*, 44(4):585–591, 1997.

[28] E. G. Talbi and P. Bessiere. A parallel genetic algorithm for the graph partitioning problem. In *Proceedings of the ACM International Conference on Supercomputing, ACM*, Cologne, 1991.

[29] Y. Weiss. Segmentation using eigenvectors: A unifying view. In *ICCV (2)*, pages 975–982, 1999.